package com.twitter.timelines.data\_processing.ml\_util.aggregation\_framework

import com.twitter.dal.client.dataset.KeyValDALDataset

import com.twitter.ml.api.DataRecord

import com.twitter.scalding.DateParser

import com.twitter.scalding.RichDate

import com.twitter.scalding\_internal.multiformat.format.keyval.KeyVal

import com.twitter.storehaus\_internal.manhattan.\_

import com.twitter.storehaus\_internal.util.ApplicationID

import com.twitter.storehaus\_internal.util.DatasetName

import com.twitter.storehaus\_internal.util.HDFSPath

import com.twitter.summingbird.batch.BatchID

import com.twitter.summingbird.batch.Batcher

import com.twitter.summingbird\_internal.runner.store\_config.\_

import java.util.TimeZone

import com.twitter.summingbird.batch.MillisecondBatcher

/\*

\* Configuration common to all offline aggregate stores

\*

\* @param outputHdfsPathPrefix HDFS prefix to store all output aggregate types offline

\* @param dummyAppId Dummy manhattan app id required by summingbird (unused)

\* @param dummyDatasetPrefix Dummy manhattan dataset prefix required by summingbird (unused)

\* @param startDate Start date for summingbird job to begin computing aggregates

\*/

case class OfflineAggregateStoreCommonConfig(

outputHdfsPathPrefix: String,

dummyAppId: String,

dummyDatasetPrefix: String,

startDate: String)

/\*\*

\* A trait inherited by any object that defines

\* a HDFS prefix to write output data to. E.g. timelines has its own

\* output prefix to write aggregates\_v2 results, your team can create

\* its own.

\*/

trait OfflineStoreCommonConfig extends Serializable {

/\*

\* @param startDate Date to create config for

\* @return OfflineAggregateStoreCommonConfig object with all config details for output populated

\*/

def apply(startDate: String): OfflineAggregateStoreCommonConfig

}

/\*\*

\* @param name Uniquely identifiable human-readable name for this output store

\* @param startDate Start date for this output store from which aggregates should be computed

\* @param commonConfig Provider of other common configuration details

\* @param batchesToKeep Retention policy on output (number of batches to keep)

\*/

abstract class OfflineAggregateStoreBase

extends OfflineStoreOnlyConfig[ManhattanROConfig]

with AggregateStore {

override def name: String

def startDate: String

def commonConfig: OfflineStoreCommonConfig

def batchesToKeep: Int

def maxKvSourceFailures: Int

val datedCommonConfig: OfflineAggregateStoreCommonConfig = commonConfig.apply(startDate)

val manhattan: ManhattanROConfig = ManhattanROConfig(

/\* This is a sample config, will be replaced with production config later \*/

HDFSPath(s"${datedCommonConfig.outputHdfsPathPrefix}/${name}"),

ApplicationID(datedCommonConfig.dummyAppId),

DatasetName(s"${datedCommonConfig.dummyDatasetPrefix}\_${name}\_1"),

com.twitter.storehaus\_internal.manhattan.Adama

)

val batcherSize = 24

val batcher: MillisecondBatcher = Batcher.ofHours(batcherSize)

val startTime: RichDate =

RichDate(datedCommonConfig.startDate)(TimeZone.getTimeZone("UTC"), DateParser.default)

val offline: ManhattanROConfig = manhattan

}

/\*\*

\* Defines an aggregates store which is composed of DataRecords

\* @param name Uniquely identifiable human-readable name for this output store

\* @param startDate Start date for this output store from which aggregates should be computed

\* @param commonConfig Provider of other common configuration details

\* @param batchesToKeep Retention policy on output (number of batches to keep)

\*/

case class OfflineAggregateDataRecordStore(

override val name: String,

override val startDate: String,

override val commonConfig: OfflineStoreCommonConfig,

override val batchesToKeep: Int = 7,

override val maxKvSourceFailures: Int = 0)

extends OfflineAggregateStoreBase {

def toOfflineAggregateDataRecordStoreWithDAL(

dalDataset: KeyValDALDataset[KeyVal[AggregationKey, (BatchID, DataRecord)]]

): OfflineAggregateDataRecordStoreWithDAL =

OfflineAggregateDataRecordStoreWithDAL(

name = name,

startDate = startDate,

commonConfig = commonConfig,

dalDataset = dalDataset,

maxKvSourceFailures = maxKvSourceFailures

)

}

trait withDALDataset {

def dalDataset: KeyValDALDataset[KeyVal[AggregationKey, (BatchID, DataRecord)]]

}

/\*\*

\* Defines an aggregates store which is composed of DataRecords and writes using DAL.

\* @param name Uniquely identifiable human-readable name for this output store

\* @param startDate Start date for this output store from which aggregates should be computed

\* @param commonConfig Provider of other common configuration details

\* @param dalDataset The KeyValDALDataset for this output store

\* @param batchesToKeep Unused, kept for interface compatibility. You must define a separate Oxpecker

\* retention policy to maintain the desired number of versions.

\*/

case class OfflineAggregateDataRecordStoreWithDAL(

override val name: String,

override val startDate: String,

override val commonConfig: OfflineStoreCommonConfig,

override val dalDataset: KeyValDALDataset[KeyVal[AggregationKey, (BatchID, DataRecord)]],

override val batchesToKeep: Int = -1,

override val maxKvSourceFailures: Int = 0)

extends OfflineAggregateStoreBase

with withDALDataset